

Práctica 1

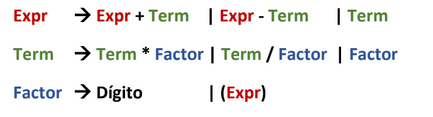
Abel Eduardo Robles Lázaro

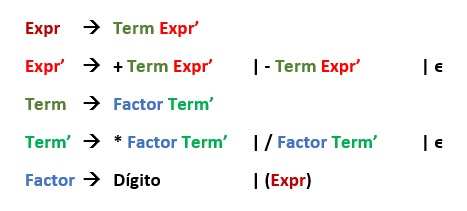
27 de febrero de 2023

Universidad de guadalajara

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

## **Introducción:**

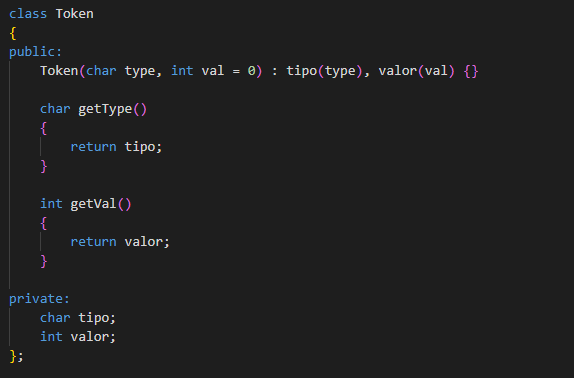
Para el desarrollo de la práctica requerimos usar las reglas de producción utilizadas en la tarea #5:

Sin embargo, éstas reglas cuentan con recursividad por la izquierda, lo que causa ambigüedad e imposibilita su implementación con un analizador sintáctico de este tipo, por lo que fue necesario eliminar la recursividad a la izquierda, adaptando las reglas de la siguiente manera:

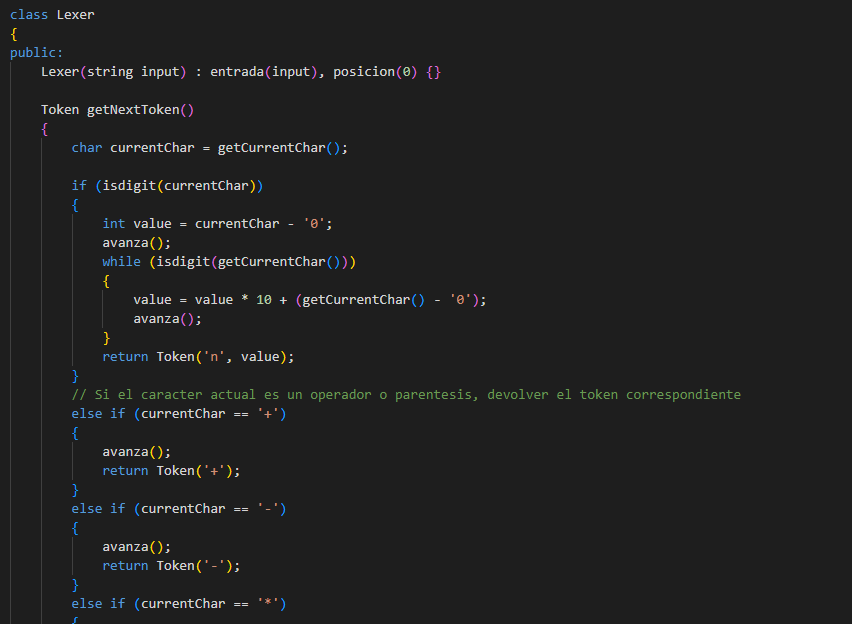
De este modo, ya sin recursividad fue posible comenzar el desarrollo de la práctica 1

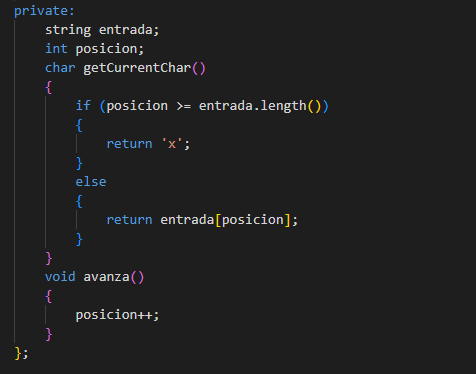
## **Desarrollo:**

Decidí seguir un desarrollo por etapas aunque no estaba seguro de lo que debía incluir el programa, ya que las especificaciones de la práctica no son precisas. Primero decidí utilizar el lenguaje C++ porque es el lenguaje con el que tengo mayor experiencia.

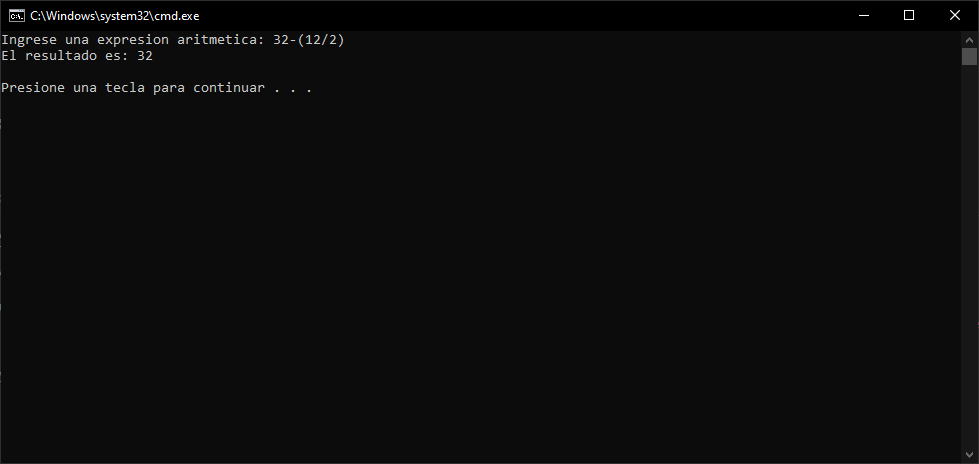
Hice una clase Token para administrar los caracteres individuales de la cadena de entrada

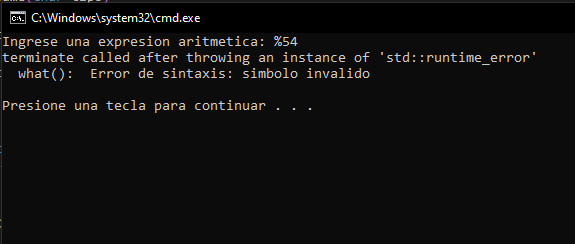
Donde el tipo indica si es un símbolo cuál es (‘+’, ‘-‘, ‘\*’, ‘/’, ‘()’) o si se trataba de un número el carácter sería ‘n’ y su valor sería almacenado en el espacio correspondiente.

Después creé la clase “Lexer” para procesar la cadena y almacenar el número adecuado en caso necesario en la clase Token.

Ésta clase también se encarga de almacenar la posición del token relativa a la cadena y avanzar sobre ésta:

La clase “Parser” es donde tuve más dificultades pues no estuve seguro de cómo traducir la jerarquía de las reglas de producción en el código:

Intenté seguir el orden predeterminado y realizar las operaciones correspondientes pero el código tiene un error en alguna parte dentro de ésta clase que no pude identificar, por lo que el código no realiza las operaciones necesarias.

Lo que sí hace correctamente es identificar cuando una cadena no es válida:

## **Código fuente:**

**Main.cpp:**

#include <iostream>

#include <string>

**using** **namespace** std**;**

**class** Token

**{**

**public:**

Token**(char** type**,** **int** val **=** 0**)** **:** tipo**(**type**),** valor**(**val**)** **{}**

**char** getType**()**

**{**

**return** tipo**;**

**}**

**int** getVal**()**

**{**

**return** valor**;**

**}**

**private:**

**char** tipo**;**

**int** valor**;**

**};**

**class** Lexer

**{**

**public:**

Lexer**(**string input**)** **:** entrada**(**input**),** posicion**(**0**)** **{}**

Token getNextToken**()**

**{**

**char** currentChar **=** getCurrentChar**();**

**if** **(**isdigit**(**currentChar**))**

**{**

**int** value **=** currentChar **-** '0'**;**

avanza**();**

**while** **(**isdigit**(**getCurrentChar**()))**

**{**

value **=** value **\*** 10 **+** **(**getCurrentChar**()** **-** '0'**);**

avanza**();**

**}**

**return** Token**(**'n'**,** value**);**

**}**

**else** **if** **(**currentChar **==** '+'**)**

**{**

avanza**();**

**return** Token**(**'+'**);**

**}**

**else** **if** **(**currentChar **==** '-'**)**

**{**

avanza**();**

**return** Token**(**'-'**);**

**}**

**else** **if** **(**currentChar **==** '\*'**)**

**{**

avanza**();**

**return** Token**(**'\*'**);**

**}**

**else** **if** **(**currentChar **==** '/'**)**

**{**

avanza**();**

**return** Token**(**'/'**);**

**}**

**else** **if** **(**currentChar **==** '('**)**

**{**

avanza**();**

**return** Token**(**'('**);**

**}**

**else** **if** **(**currentChar **==** ')'**)**

**{**

avanza**();**

**return** Token**(**')'**);**

**}**

**else**

**{**

**throw** runtime\_error**("Error de sintaxis: simbolo invalido");**

**}**

**}**

**private:**

string entrada**;**

**int** posicion**;**

**char** getCurrentChar**()**

**{**

**if** **(**posicion **>=** entrada**.**length**())**

**{**

**return** 'x'**;**

**}**

**else**

**{**

**return** entrada**[**posicion**];**

**}**

**}**

**void** avanza**()**

**{**

posicion**++;**

**}**

**};**

**class** Parser

**{**

**public:**

Parser**(**Lexer **&**lexer**)**

**:** lexer\_**(**lexer**),** currentToken\_**(**lexer**.**getNextToken**())**

**{**

**}**

**int** expresion**()**

**{**

**int** result **=** termino**();**

**while** **(**currentToken\_**.**getType**()** **==** '+' **||** currentToken\_**.**getType**()** **==** '-'**)**

**{**

**if** **(**currentToken\_**.**getType**()** **==** '+'**)**

**{**

consume**(**'+'**);**

result **+=** termino**();**

**}**

**else**

**{**

consume**(**'-'**);**

result **-=** termino**();**

**}**

**}**

**return** result**;**

**}**

**int** termino**()**

**{**

**int** result **=** factor**();**

**while** **(**currentToken\_**.**getType**()** **==** '\*' **||** currentToken\_**.**getType**()** **==** '/'**)**

**{**

**if** **(**currentToken\_**.**getType**()** **==** '\*'**)**

**{**

consume**(**'\*'**);**

result **\*=** factor**();**

**}**

**else**

**{**

consume**(**'/'**);**

result **/=** factor**();**

**}**

**}**

**return** result**;**

**}**

**int** factor**()**

**{**

**if** **(**currentToken\_**.**getType**()** **==** '('**)**

**{**

consume**(**'('**);**

**int** result **=** expresion**();**

consume**(**')'**);**

**return** result**;**

**}**

**else** **if** **(**currentToken\_**.**getType**()** **==** 'n'**)**

**{**

**int** result **=** currentToken\_**.**getVal**();**

consume**(**'n'**);**

**return** result**;**

**}**

**else**

**{**

**throw** runtime\_error**("Factor invalido");**

**}**

**}**

**private:**

**void** consume**(char** tipo**)**

**{**

**if** **(**currentToken\_**.**getType**()** **==** tipo**)**

**{**

currentToken\_ **=** lexer\_**.**getNextToken**();**

**}**

**else**

**{**

**throw** runtime\_error**("Token inesperado: "** **+** currentToken\_**.**getVal**());**

**}**

**}**

Lexer lexer\_**;**

Token currentToken\_**;**

**};**

**int** main**()**

**{**

string input**;**

cout **<<** **"Ingrese una expresion aritmetica: ";**

getline**(**cin**,** input**);**

Lexer lexer**(**input**);**

Parser parser**(**lexer**);**

**try**

**{**

**int** resultado **=** parser**.**expresion**();**

cout **<<** **"El resultado es: "** **<<** resultado **<<** endl**;**

**}**

**catch** **(const** exception **&**ex**)**

**{**

cerr **<<** **"Error: "** **<<** ex**.**what**()** **<<** endl**;**

**}**

**return** 0**;**

**}**

## **Conclusiones:**

Éste programa aún tiene mucho trabajo por delante, espero recibir indicaciones más detalladas para la próxima ocasión y así desarrollar un programa funcional.